Anthony Martinez
CSCI 499
2524464843
Project Paper
Paper: "Two Theorems on Random Polynomial Time"

On Adleman's "Two Theorems on Random Polynomial Time"

Leonard Adleman's "Two Theorems on Random Polynomial Time" was published in the "19th Annual Symposium on Foundations of Computer Science" in 1978. The paper was written during his time at the Massachusetts Institute of Technology (MIT). Essentially, Adleman introduced two theorems dealing with the class R (also known as RP[1]), the class of languages decided by a probabilistic[2] polynomial time Turing machine. In the introduction to the paper, Adleman notes the uses of randomness in demonstrating whether problems are in the class P or not (tractable v. intractable) as well as the applications of the theorems to Circuit Complexity to establish or refute that NP ≠ P. Furthermore, Adleman predicts that the two theorems introduced in the paper will help develop stronger methods that work to demonstrate how to classify the intractability of problems such as linear programming and graph isomorphism (which have not yet been shown to be NP-complete[3]). Adleman acknowledged the advances that randomness in computation brought to computer science research and set out in his paper to conduct a new examination on the nature of randomness.

In Theorem I[4], Adleman shows that all languages in R have polynomial sized circuits[5]. Theorem I provides an avenue for proving that NP ≠ P by analyzing the circuit complexity of problems in NP. If one shows that there is a problem in NP that does not have polynomial sized circuit complexity, then you also show that NP ≠ P based on results[6] we have shown relating circuit to time

[1] See Glossary for definition
[2] See Glossary for definition
[3] Citation needed
[4] See Glossary
[5] Circuit complexity is defined as the minimum number of gates required to decide the language
[6] From Circuit-SAT, if a language A has runtime $O(f(n))$ then it has circuit complexity $O(f^2(n))$

complexity. Adleman states that his result that all of R has polynomial sized circuits does not suggest that one cannot take this route to show NP ≠ P, but rather that if one were to take this method, two considerations are required. First, generalizations of Theorem I suggest that classes "close[7]" to P are most understood[8] and therefore most likely to not have large circuit complexity (i.e. Possible to calculate lower bounds). The second consideration is the consequence resulting from the proof of Theorem I that the set of Random-complete problems has non-polynomial circuit complexity if and only if NP-complete ones also have (Adleman). This result establishes that showing that SAT (or any NP-Complete problem) has non-polynomial size circuits, that Random-complete problems are not in P. These two considerations are among many in using the new techniques proposed by Adleman involving Random-complete sets.

The proof of Theorem I involves a counting argument which tracks that there are at most n "witnesses[9]" that are valid. The bounding n implies that the size of the circuit is polynomial. The construction uses a tableau (or matrix) of the "z" elements in a language $A \in R$ that are of size less than or equal to n in rows and the witnesses for such elements of A as columns (of which there are n). The proof uses iterations to destruct the tableau. The idea is that since A is in R, at least half of the entries in the matrix are "ones" (accepting answers, of which there are $\frac{z*n}{2}$). There are only n witnesses (columns), implying that there is a witness $w_1$with at least half ($\frac{z}{2}$) of its rows as "ones" (indicating an accepting result). The iteration removes from the tableau that column and rows that lie on the tableau having a "one" for that w_1 column to construct a new tableau (matrix). That matrix also has at least half of its entries as "ones". This

[7] A good definition of a class can be considered "close" to P is if it is the class of decidable problems solvable in polynomial time outputting a wrong answer only on a sparse set of instances (Yap). Indeed, this makes sense as the paper later shows that $R \subseteq P/poly$ and it is the case that P-Close is contained in P/poly (Schöning)

[8] Circuit size lower bounds are difficult to show or prove; usually results of lower bounds are shown for well understood problems

[9] Verifier

deconstruction is repeated v times, resulting in $v \leq n$ and $\forall a \in A\ \exists j \leq v\ s.t.\ w_j\ accepts/witnesses\ a$.

Theorem II[10] demonstrates a consequence immediate from showing that NP ≠ R. It states that following such a result, there must exist a language in NP that is R-hard[11] and not NP-complete[12]. While very intuitive, the proof[13] of Theorem II is complex and involves using a "Ladner-like construction" (Ladner).

While Theorem II as stated is not rather useful in terms of showing properties of the class R and NP, an alternative variant of Theorem II shows that for $\Delta_R = R \cap Co-R$, if $\Delta_R \neq NP$ then there is a language $A \in NP$ such that A is $\Delta_R - hard$[14] and A is not NP-complete (Adleman). Again, this result is rather intuitive. This alternative result can be used to classify problems, such as graph isomorphism, that have not been shown to be NP-complete. Consensus seems to indicate that Δ_R is quite possibly not equal to P. $\Delta_R \neq P$ intuitively implies that R-hard problems are not in P. Theorem II is applicable when considering that it may be the case that a problem such as graph isomorphism is not sufficient to encode (be reducible to) every problem in NP but can still encode every problem in Δ_R. To use Theorem II to show a problem is $\Delta_R -$ hard, one would have to find a special property of languages in Δ_R that distinguishes them from other languages in NP. This is where Theorem I comes into play, which proved properties about Δ_R circuit complexity being of polynomial size.

A lasting implication of Adleman's Theorems is the result that the class BPP[15] is contained in P/poly. Theorem I shows that all languages in R have polynomial sized circuit complexity. P/poly is the set of problems solvable in

[10] See Glossary
[11] Language A is R-hard if $\forall B \in R,\ B \leq_p A$
[12] Language A is NP-complete if $A \in NP\ and\ \forall B \in NP, B \leq_p A$
[13] See Appendix for an informal summary of Adleman's proof of Theorem II
[14] Language A is $\Delta_R - hard\ iff\ \forall B \in \Delta_R, B \leq_p A$
[15] See Glossary for definition

polynomial time with a polynomial length advice string[16]. We have the result[17] that polynomial sized circuits exactly solve P/poly (Cote 25). Adleman's Theorem I shows that the class R has polynomial sized circuit complexity. P/poly contains all problems solvable in randomized polynomial time. With these two results, we have that $R \subseteq P/poly$. Shortly after in 1981, work done by Charles Bennett and John Gill generalized Adleman's result to show that $BPP \subseteq P/poly$ (Wikipedia_P/poly) (Bennett and Gill). P/poly itself is a very interesting class, with several important properties depending on its relationship with other complexity classes. Both theoretical and practical such as the Karp-Lipton theorem[18] on the polynomial hierarchy[19] if $NP \subseteq P/poly$ and rainbow table[20] cryptography using the tables as advice strings respectively.

Adleman's paper proposed two theorems on the class R and its relationship with the class NP and the uses of the theorems on showing the relationships of other complexity classes. Specifically, Adleman proposed Theorem I proving that the class R has polynomial sized circuits and Theorem II that if $NP \neq R$ that there's a problem A in NP that is R-hard and not NP-Complete. Subsequent research has shown results using both theorems that the class P/poly is directly solved by problems of polynomial sized circuits, implying that the class R is contained in P/poly. This result has been used in areas such as cryptography and complexity properties in terms of hierarchies. Adleman's paper was among many at the time and thereafter that attempted to solve and categorize hierarchies using properties about the class R. While

[16] The advice string depends on the length of the input, not the input itself
[17] Informally, there is a different circuit C_i that decides an element of language A for each length of the input. This C_i can be thought of as the specific advice strings for different length inputs. As a result, polynomial sized circuits exactly solve P/poly.
[18] The Karp-Lipton theorem simply states that the polynomial hierarchy would collapse to its second level if $NP \subseteq SIZE\left(n^{O(1)}\right)$
[19] Simply put, it is an oracle Turing machine generalization of the classes P, NP, and co-NP (Wikipedia).
[20] A rainbow table is a precomputed table for reversing cryptographic hash functions (to recover the original input for instance), this can be thought of as the advice string to a P/poly set

many questions remain, thanks to him we inferred many properties about classes related to R.

Glossary

The following terms and definitions are cited from the Sipser textbook unless otherwise cited

1. **Theorem I** (Adleman):
$$\text{For all } A \in R,$$
$$A \text{ has polynomial size circuits}$$
2. **Theorem II** (Adleman):
$$\text{If } NP \neq R$$
$$\text{then there is a set } A \in NP \text{ such that:}$$
$$\forall B \in R, B \leq_p A$$
$$SAT \nleq_p A$$

3. **Probabilistic Algorithm:** An algorithm designed to use the outcome of a random process. Typically, such an algorithm would contain an instruction to "flip a coin" and the result of that coin flip would influence the algorithms subsequent execution and output. (Sipser) It takes advantage of the idea that it might be to computationally hard to calculate or even estimate the best choice in a situation.
4. **Probabilistic Turing Machine:** a type of non-deterministic Turing machine in which each non-deterministic step is called a *coin-flip step* and has two legal next moves. We assign a probability to each branch b of M's computation on input w as follows. Define the probability of branch b to be:
$$\Pr[b] = 2^{-k},$$
Where k is the number of *coin-flip steps* that occur on branch b
Define the probability that M accepts w to be
$$\Pr[M\ accepts\ w] = \sum_{b\ is\ an\ accepting\ branch} \Pr[b]$$

Essentially, the probability that the machine accepts w is the probability that we would reach an accepting configuration if we simulated M on w by slipping a coin to determine which move to follow at each coin-flip step.

When a probabilistic Turing machine decides a language, it must accept all strings in the language and reject all strings out of the language as usual, except now we allow the machine a small probability of error. (Sipser)

5. **BPP:** The class of languages that are decided by probabilistic polynomial time Turing machines with an error probability of 1/3. (Sipser)
6. **RP:** The class of languages that are decided by probabilistic polynomial time Turing machines where inputs in the language are accepted with a probability of at least 1/2 , and inputs not in the language are rejected with a probability of 1 (Sipser)

Appendix

Summary of Theorem II Proof (Adleman)

Informal proof paraphrased where acceptable shown below for clarity.

- $\{P_i\}$ is defined as the usual recursive enumeration of polynomial time deterministic Turing machines. A recursive enumeration if one recalls is simply an algorithm that enumerates members of the language.
- $P_i\ is\ defined\ as\ the\ \pi_1(i)\ the$ deterministic Turing machine with a $n^{\pi_2(i)}$ clock
- π_1, π_2 are defined to be projection functions associated with a pairing function "$< >$". A projection function is essentially a function U_i^n that extracts the ith coordinate from an ordered n-tuple (i.e. Ordered pair). A pairing function is a process to uniquely encode two natural numbers into a single natural number, i.e. it can show that two sets have the same cardinality. (Wikipedia)
- $\{M_i\}$ is defined as a recursive enumeration of polynomial time non-deterministic Turing machines
- M_i is defined as the non-deterministic Turing machine which on input X guesses all strings Y of length $\leq\ |X|^{\pi_2(i)}$ and accepts if and only if $P_{\pi_1(i)}(\langle x, y\rangle) = 1$
- M_i is an R-Machine if and only if $\#\ accepting\ paths \geq \frac{1}{2}\ \forall X\ accepted\ by\ M_i$
- Set of R-Machines is not recursively enumerable. Class R is recursively enumerable.
- R-Machines are $co-RE$[21] $\Rightarrow\ \exists$ an algorithm that recognizes M_i that are not R-Machines by brute force searching for an input x which is accepted on $\leq \frac{1}{2}$ the paths.

For any pair i, j, let $\phi_{i,Y}$ be the formula given by the standard Cook construction

$$\text{Thus, } M_i\ accepts\ y\ \leftrightarrow \phi_{iY} \in SAT$$

$$\text{Let } f(x) = \begin{cases} \langle i, Y\rangle\ if\ x = \phi_{i,Y}\ for\ some\ i\ and\ Y \\ 0\ otherwise \end{cases}$$

i.e. $f(x)$ returns the value from the pairing function if the assignment x matches $\phi_{i,Y}$ or zero otherwise.

$f(x)$ is computable in polynomial time since the Cook construction produces outputs $\phi_{i,Y}$ which are syntactically transparent and i and Y can easily be read

[21] Co-RE refers to the set of all languages that are complements of a language in RE, the class of languages for which there is a verifier but not necessarily a decider

The following construction will take in input x and will produce the set A by using an oracle for SAT using deterministic polynomial time algorithm Df for the function $f(x)$ above and deterministic (non-polynomial time) algorithms DSAT and DA for deciding SAT and A respectively.

1. For $|x|$ steps:
   a. Using DSAT, DA, and P_1, exhaustively search for a Y such that $Y \in SAT$ and $P_1(Y) \notin A\ or\ Y \notin SAT\ and\ P_1(y) \in A$. That is, find a counter example to SAT being polytime (Karp) reducible to A ($SAT \leq_{P_1} A$).
   b. If you found such a Y, then repeat a. for $P_2, P_3, \dots, P_{|x|}$. That is, repeat until a total of |x| steps are exhausted
   c. Let j be the least number such that no counter example for P_j was found.
2. For |x| steps:
   a. For $0 \leq i \leq j$, form a deterministic search for a Y such that running M_i on Y accepts on less than $\frac{1}{2}$ of its paths. This proves that M_i is not an R-machine based on our definition of an R-machine.
   b. Let S_X be the subset of $\{0,1,\dots,j\}$ for which no such proof was found. Specifically, S_X is the set of indices of machines which could still be R-machines

This completes the proof. Phase 1 of the construction showed that A is not NP-Complete since we found at most j counter examples for the reduction. Phase 2 of the construction showed that A is still in R and is in fact R hard.

References

Adleman, Leonard. "Two Theorems on Random Polynomial Time." *19th Annual Symposium on Foundations of Computer Science* (1978): 75-80. <https://ieeexplore.ieee.org/document/4567965/references#references>.

Aspens, James. "Notes on Randomized Algorithms." Yale, February 2018. <http://www.cs.yale.edu/homes/aspnes/classes/469/notes.pdf>.

Bennett, Charles H and John Gill. "Relative to a Random Oracle A, PA != NPA != co-NPA with Probability 1." *SIAM J. Comput.* (1981): 96-113. <http://www.cs.cornell.edu/courses/cs682/2006sp/Handouts/BennettGill.pdf>.

Cote, Aaron. "CSCI 499 Theory of Computation." n.d. Lecture.

Ladner, Richard. "On the Structure of Polynomial Time Reducibility." *Association for Computing Machinery* (1975): 155-171.

Schöning, Uwe. "Complete sets and closeness to complexity classes." *Mathematical Systems THeory* 19.1 (2985): 29-41.

Sipser, Michael. *Introduction to the Theory of Computation*. 3rd. Cengage Learning, 2012.

Trevisan, Luca. "Adleman's Theorem." *Handout 4*. Stanford University, 7 April 2010. April 2019. <https://www.cs.stanford.edu/~trevisan/cs254-10/lecture04.pdf>.

Wikipedia_P/poly. *P/poly*. 11 December 2018. Wikipedia, The Free Encyclopedia. Web. 18 April 2019. <https://en.wikipedia.org/w/index.php?title=P/poly&oldid=873124771>.

Yap, Chee K. "Some consequences of non-uniform conditions on uniform classes." *Theoretical Computer Science* 26.3 (2983): 287-300.